API Designers in the Field: Design Practices and Challenges for Creating Usable APIs

Conference Paper · October 2018

DOI: 10.1109/VLHCC.2018.8506523

Brad A. Myers
Carnegie Mellon University

API Designers in the Field: Design Practices and Challenges for Creating Usable APIs

Lauren Murphy
University of Michigan
laumurph@umich.edu

Mary Beth Kery
HCII, CMU
mkery@cs.cmu.edu

Oluwatosin Alliyu
Haverford College
alliyut@gmail.com

Andrew Macvean
Google, Inc.
Seattle, WA
amacvean@google.com

Brad A. Myers
HCII, CMU
bam@cs.cmu.edu

***ABSTRACT*—Application Programming Interfaces (APIs) are a rapidly growing industry and the usability of the APIs is crucial to programmer productivity. Although prior research has shown that APIs commonly suffer from significant usability problems, little attention has been given to studying how APIs are designed and created in the first place. We interviewed 24 professionals involved with API design from 7 major companies to identify their training and design processes. Interviewees had insights into many different aspects of designing for API usability and areas of significant struggle. For example, they learned to do API design on the job, and had little training for it in school. During the design phase they found it challenging to discern which potential use cases of the API users will value most. After an API is released, designers lack tools to gather aggregate feedback from this data even as developers openly discuss the API online.**

Keywords—API Usability, Empirical Studies of Programmers, Developer Experience (DevX, DX), Web Services.

I. INTRODUCTION

An Application Programming Interface (API) describes the interface that a programmer works with in order to communicate with a library, software development kit (SDK), framework, web service, middleware, or any other piece of software [1]. Given their ubiquity, APIs are tremendously important to software development. With the rise of web services, APIs are projected to rapidly grow into a several hundred billion dollar industry [2] [3]. APIs are increasingly a primary way that businesses deliver data and services to their user-facing software and also to client software at other companies that purchase and receive that business's services via APIs [2] [4]. As of April, 2018, programmableweb.com listed over 19,500 APIs for Web services, with hundreds more being added each year.

The study of API Usability [5] has often focused on the programmers who use APIs in their own code, known as *API Users*. The developer experience (DX or DevX) is significantly impacted by the quality of the APIs they must use. However, the usability that these programmers experience with an API ultimately stems from how well the API and its resources were designed in the first place. Very little attention so far has been given to the API design processes or to the *API designers* who are making these design decisions. With an increasing number of companies creating APIs, it is important to have an understanding of API design in the field. What is the current process of designing and producing APIs in real organizations? What current roles do human-factors and usability play in design decisions? What are the open challenges that API designers face?

In order to better understand the real-world design process of APIs and challenges of API designers, we conducted interviews with 24 professional software engineers and managers experienced in API design across 7 major companies. We found interesting insights about how APIs are designed and ways to improve this process. For example, we found that API design is learned most often through practice and can be cultivated through exposure to API design reviews. These reviews preferably occur at multiple stages in API development. Early feedback from users on an API's design and future use cases will result in a better design, but was reported to be challenging to obtain. User testing - even quick, informal studies - can be greatly beneficial. Documentation that is not discoverable can turn away potential customers. More customization is needed for general-purpose documentation and SDK generators but success has been found with custom ones. Other findings are discussed throughout.

II. RELATED WORK

A. API Usability

Since APIs are a form of user interface, considering the usability of that interface is important [6]. API usability has been shown to impact the productivity of the programmer, the adoption of the API, and the quality of the code being written [1]. If used incorrectly, research shows that the resultant code is likely to contain bugs and security problems [7].

However, APIs are often hard for programmers to learn and use [1], with prior work identifying many causes, including the semantic design of the API, the level of abstraction, the quality of the documentation, error handling, and unclear preconditions and dependencies [8]–[13]. Additionally, changing an API after it has been deployed is difficult, due to its potential to break the software that depends on it [14]. All of this combines to make the API design process critically important.

B. API Design

There are a number of decisions API designers must make to create an API [15], and quality attributes that must be

978-1-5386-4235-1/18/$31.00 ©2018 IEEE

evaluated [1]. The impact of some API design decisions have been explored, providing API designers with empirically based guidelines to follow. For example, the use of the factory pattern [16] and required constructor parameters [17] were shown to be detrimental, and method placement was shown to be crucial [18]. Additionally, recent work has looked at defining metrics for encapsulating and measuring API usability, allowing for larger scale quantitative assessment of an APIs design [19], [20]. In some cases, as much as 25% of the variance in the proportion of erroneous calls made to an API could be predicted by modeling just 7 structural factors of the API's design, including the number of required parameters in the API call, and the overall size of the API [21]. On this vein, several recent research projects are exploring new forms of static analysis tools to detect API usability issues [41].

To collect and standardize common API design decision, a number of major companies like Google [22] and Microsoft [23] have released API Style Guides. A recent review of these guides identified both core principles, and inconsistencies contained their advice to designers [24]. Finally, although online API style guides are the most up-to-date resource for API designers, a number of useful principles for API design can be found in software engineering books [25][26], as well as classic theories for evaluating usability, such as the Cognitive Dimensions of Notation [27].

C. *Understanding The API Design Process*

There are a number of examples of the positive impact that can result from using traditional HCI methods during the API design process, including usability studies, design reviews, and heuristic evaluations to aid in understanding and improving the APIs' usability [27]–[31]. However, more broadly understanding the needs of the API designers, and the process they go through while they design, implement, release, and maintain an API, is less well explored. Macvean et al. discuss part of the web API design process at Google, which includes an expert review of a proposed API design. The goal is to ensure consistency and quality while providing API designers with the ability to consult with API design experts [28]. Henning stresses the importance of education for successful API design, arguing that good API design can be taught [32].

While there has been much work done in understanding the software engineering process in general, e.g., the role of knowledge sharing within organizations [33], the impact of distance between engineers [34], and the importance of well defined tasks [35], understanding the specifics of the API design process, and the barriers facing API designers, have not been previously studied.

III. METHODOLOGY

To better understand the processes, challenges, and constraints of API designers, we conducted structured interviews with practicing API designers from industry. First, we iteratively constructed 36 questions through discussion with several of our current and former collaborators who have API research or qualitative research experience. The resulting questions (many of which had follow-up subparts) focussed on a broad range of API design topics, including what a good API is, the design processes they follow, difficulties faced when designing, and how to improve this process in the future. (The full questionnaire is available in the supplemental material and at *natprog.org/papers/InterviewScript.pdf.*) We interviewed 24 different API designers from seven large companies, from various positions within the companies. Each interview lasted from 60 to 120 minutes, and was audio recorded which was complemented with detailed notes. The interviews were all performed via video conferencing, since the interviewees were remote (with participants from USA, Asia, and Europe). All the interviews took place during working hours, and there was no compensation offered.

We transcribed a total of 38 audio hours of interview data for our analysis.. For two interviews where the audio failed or was too poor quality to transcribe, we relied on the detailed notes for analysis. Following grounded theory methodology [36], the 1st author first sampled four interviews to do open-coding which generated 505 codes. Those codes were reduced down to 41 specific labels, such as Usability Factors, Company Processes for API Lifecycle, and Audience & Use Cases. At first, we followed standard inter-rater reliability [37] as we expected the priorities of the designers to be fairly consistent. However, we found a rich diversity from the designers, and thus, the use of inter-rater reliability proved to be a misstep. Initially, the 1st and 2nd author each independently coded a new sample of five analyses (20% of the data), receiving a low Cohen's Kappa of 0.55 [37]. Both authors discussed disagreements, refined the code book, and repeated the process on a new sample of five interviews. With a moderate Cohen's Kappa of 0.71 [37], the two authors labeled all remaining interviews together, allowing for multiple labels where needed, as decided through discussion and consensus. Afterwards, our analysis followed 'data-driven' thematic analysis [42] where we clustered our coded data into themes.

IV. PARTICIPANTS

We used snowball sampling to recruit: first inviting personal contacts to participate, and they in turn asked appropriate people at their companies who were involved in API Design. Between June 2017 and February of 2018 we interviewed 24 participants from seven large companies: one financial company and six tech companies. Participants had an average of 16 years programming experience, with an average of 9 years API design experience. Five participants are currently in upper management, while all others were software engineers. Participants will be called P1 to P24. Quotes below are filtered to protect anonymity, and instances where a participant mentions company-specific terms will be replaced by "X". Table 1 summarizes the types of APIs that participants worked

on. Note that 9 of 24 worked on more than one type. All had designed APIs that were in use by many programmers, ranging from hundreds to millions of users.

TABLE 1. TYPES OF APIs THAT PARTICIPANTS WORKED ON

REST API	P1, P2, P3, P4, P5, P6, P7, P15, P17, P18, P19, P20, P21, P22, P23, P24
Other Web API	P1, P5, P6, P7, P8, P16, P17, P18, P21, P23
Other Library or SDK API	P6, P9, P10, P11, P12, P13, P14, P19

TABLE 2. AUDIENCE OF THE APIs

Public API	P1, P2, P3, P4, P6, P7, P8, P9, P11, P12, P14, P17, P19, P20, P21, P22, P24
Internal API	P1, P5, P6, P10, P12, P13, P15, P18, P20, P23, P24
Gated API	P15, P16, P17, P22

In Table 2, Public APIs are those available to any developer to use. Internal APIs are built in-house and used only within a company. Gated APIs have a select set of enterprise customers who directly communicated their feedback to the API team.

V. RESULTS

A. Learning to Design APIs

All API designers we talked to had primarily learned to design APIs through work experience. Raw experience with designing APIs and learning from colleagues was prized over any form of formal educational resource. The designers' education level ranged from a PhD to a high-school diploma. Some had taken software engineering courses, yet only four participants had learned anything about API Design in school.

Even when interviewees came from large software companies, they reported that API design was a specialization of a relatively small group of people in their company. API design experts are relatively rare. We asked participants what would differentiate a good API designer from a regular software designer. API designers must have a strong understanding of software engineering, but *also* have the ability and personal drive to stay focused on their user's perspective as their primary goal:

"*A good API designer would put [themselves] in the shoes of [another] person who is actually going to use the API whereas a good software designer would basically look at it from [their own] perspective if the design is good or whether the design is scalable*." - P21

As a solution to foster more experts, five participants reported that their company had a mentorship program, in which novice API designers (typically any engineer interested in the topic) could sit in on the expert API design reviews.

Implications: API design is recognized as a difficult and specialized skill with few training resources, so more training material is needed. On the job, the experience that expert designers need could be scaffolded much like many other design disciplines are taught, with creation and critique exercises. Novice API designers would practice creating APIs (plausible design exercises that are not on the critical path of a real product) and have their work critiqued in API design reviews. Additionally, since a good API designer needs an intuition for user experience, this may be trained by giving interested developers exposure to basic user experience testing and usability methods that have been well established in UX.

B. Using Existing Guidelines

Today, API guidelines published online by various organizations are the primary authoritative source for design standards and practices [24]. Many companies look to and copy from API guidelines of industry leaders, consistent with observations from Murphy et al. [24]. Nine participants we spoke with were active contributors to API guidelines at their organization. Although they might use the principles from the API guidelines, not all designers we spoke to had actually read the entirety of their company's guidelines, instead using it as a general-purpose reference material to look up specific design questions as they arose.

However, some participants found that the broad nature of the guidelines left them still needing guidance when it came to specific decisions. Participants disagreed about whether the guidelines they had were sufficient to really serve as a design tool or if and how they should be improved. This disagreement often centered around the inclusion of recommendations specific to use cases. Five participants reported making custom guidelines for their team's use cases to supplement the company-wide guidelines.

Some companies enforced their API guidelines through the use of code reviews and "linter" tools that check for specific guideline requirements. The purpose of this enforcement is to ensure a base level of API usability across the company, and also ensure API users have a *consistent* experience if they use multiple APIs across the company to avoid each API having its own learning curve. When a team or company did not systematically enforce the guidelines through a built-in part of code review culture or linter tools, adherence was difficult:

"When we got acquired by [Company X], *and we found out about this* [Company X] *standard, I was actually relieved… It was very hard [before that] to get our service engineers to consistently represent certain concepts exactly the same way. And the* [Company X] *REST standard just laid that out."* - P24

Finally, as discussed further below, designers often have multiple competing design concerns and constraints to balance during API design, making following *all* guidelines much harder than it might appear. Specifically, new designers struggle with knowing the relative importance of different guidelines, and must learn through mistakes when to adhere or deviate from a rule. One participant suggested that examples and case studies of specific guidelines in the API would help

them learn how and when to apply them. Though no other participants mentioned case studies specifically, four others wished for better rule-by-rule rationale to be included.

Implications: Best practices include following API design guidelines, which might be locally defined or adapted from other companies. At their core, design guidelines are collected wisdom from many different developers over time, so that repeated design decisions do not need lengthy discussion every time, and so that informed decisions can be made about when to break from convention. Company-wide and industry-wide API guidelines are not a one-size-fits-all rulebook, and where needed developers must supplement with team-specific or product-specific guidelines. By documenting their problem-specific API design decisions, a team can help achieve *consistency* in their future design decisions, which is a core tenant of how to make APIs (and any other form of user-interface) more easily learnable by users.

C. *Who designs an API?*

When asked what roles in their organization are involved in API design, designers said new APIs are typically requested from upper management or solution architects and first specified by project managers. More rarely, an engineer or product manager could propose an API be created, and some companies had pathways where that person could write up a proposal for the higher management to approve.

"*So there's a theoretical view that says the offering managers or the product managers are the ones who are deciding what the functionality should be, and the API designers and the engineering team in general are just deciding how to convey that functionality how to present it. But in reality the two are much more closely constrained.*" - P2

When an API's domain of use is not a highly technical engineering domain, such as providing product data or financial data, key design decisions come first from a product manager. Highly technical engineering APIs, such as those whose target users are database or network engineers, are more heavily designed by engineers. Although three participants mentioned trying to involve user experience (UX) experts or technical writers in the process to help choose good abstractions and naming, the technical knowledge needed to design for code developers is a major barrier:

"*The big problem with bringing UX people in is that they don't have a background in APIs or [even], in some cases, in programming. ...They get overwhelmed by either the people or what it is that we're putting in front of them*" - P1

Thus when it comes to usability concerns around the developer experience of what specific code the API users will need to write, the brunt of responsibility falls on the company's software engineers to seek understanding of and design for their users.

Implications: Whomever designs the API is responsible for figuring out their API users' needs and keeping those needs central to the design process. Ideally, API design should be done by an interdisciplinary team, however the challenge remains of teaching expert software engineers enough user-centered design and teaching expert UX designers enough software engineering that these two disciplines can work effectively together.

D. *Designing an API from scratch*

No participants complained of insufficient engineering expertise in their teams to specify the functions and datatypes for an API. One participant even said that they were happy to give API design tasks to a junior engineer as a learning opportunity. Rather, a major challenge for three participants was that teams commonly poured far too much time into designing and specifying *for the wrong use cases*:

"*We often spend lots of time worrying about these edge cases that in essence zero or nearly zero people end up using*" - P2

The business value of the API which is expressed at a high level like to "provide email data" or "provide access to cloud computing" or "provide a language specialized for X" is generally too abstract to anticipate the specific use cases and constraints the customer developer is going to have for an API. At the initial stage, designers aimed to release the API as quickly as possible, with a minimum amount of functionality needed to get the API product on the market. Participants said poor quality "bottom-up" API design occurs when, lacking real use case data, the engineering team designs around what is most straightforward to implement, which means that the API design mirrors much more the underlying *implementation* of the API than how customers want to use it.

"*Knowing how many people are using your API and for what, is... often difficult to do*" - P14

For internal APIs, where all users of the API are in house, the risk of getting an API wrong the first time is fairly low:

"*I start with an API that's just does the minimum possible and if it doesn't work we can change it later.*" - P9

In contrast, for publically released APIs, *all* decisions good or bad quickly become canon in users' code, since API users have been known to depend on aspects of the API as low level as the line numbers reported in error messages:

"If you change anything you basically break people ... so you need to plan much more, how should it be used, will it be used and you cannot change it afterwards so it's much much harder." - P9

It should be noted that while many types of software have to deal with updates and backwards compatibility, the case with publically released APIs is quite severe. The use of an API is baked into the API users' code, meaning that any change to the API has the potential to break the customer's code and

require many engineering hours from the customers to update their code with the new API version. To avoid "breaking changes", it is important that the API designers get core abstractions and core methods of the API correct the first time. As in any usability decision, developers must prioritize making some use cases easier than others. In an API, the core abstractions should ideally fit the real-life core use cases as closely as possible, because this permanently affects the current *and future* usability of the entire API unless serious breaking changes are possible:

"*Over-specifying things can sometimes be troublesome. For example, we've had the concept of* X *in our APIs, and it over-specified the* X *and had things in there that aren't used. ...You can't take them away because that breaks existing clients.*" - P23

Implications: Before getting too far into construction details of the API (the things that are covered in API design guidelines) like naming, pagination, etc., it is critical to check your team's understanding of the API's real life client use cases, since it will often be difficult or impossible to change these later. This can be achieved by getting users involved early on in the design process and continually obtaining feedback from them throughout this process.

E. Getting User Feedback on Initial API Design

Eighteen designers reported that they start developing their products with common use cases in mind. In the case of gated APIs developed for specific customers, designers had the ability to directly communicate with their users to understand what the use cases would be. When the APIs were meant for internal use, getting feedback about use cases was also direct:

"Most of my work have been more internal… often time we're coming in with a very better understanding of the users, we can just talk with them directly." - P5

For public APIs, (incidentally where the risk of getting an API wrong is also highest) participants most often reported that they tried to imagine themselves as future users and then built use cases largely on intuition:

"For cases where the API is new and there is essentially no usage, then obviously at that point you're relying on either your own experience as a designer or what use cases you can manage to glean from people that say, 'yes, I'd like to have a thing like that,'" - P14

Two participants mentioned creating "user stories" to base their design around. One such designer compared these user stories to personas, which are often a component in UX design. One designer said they used cognitive dimensions [27] as a method for API design to think through a user's experience. Another designer, who happened to have some training in human-computer interaction, took design ideas to informally test with any other developers in the lunchroom:

"*I was working on a design for [API X] just the other day and I was running really quick and dirty user studies in the café during lunch and it helped! I got tons of questions answered. …. I've gotten other people to do it, and … it has affected the API design. Before they put tons of time in crafting a metric name for some monitoring API, right? Like taking their candidate names in front of people and having them explain what those metrics are.*" - P1

Even though P1 worked on a public API, getting feedback from a broad range of developers inside the company (but outside of the API X team itself) gave P1 more insight about the use cases and perspectives. More generally, in order to test the ease of use of an API design, participants mentioned that it is a good sign if a developer (an API user, an API reviewer or just a convenience sample of developers in the company) is able to read through the API specification and have a good understanding of what the API does based on the names alone without relying on documentation. API Peer Reviews [31], which have someone interpret the API by names alone is also a usability test that could easily be performed early in the API design phase where only the specifications, and not the concrete implementation, exist.

Gathering use case data at this early sketching phase of designing an API was challenging to all participants. However, once the API was an implemented prototype, designers were comfortable using beta-testing and obtaining feedback that is typical of most any new software:

"We had a lot of clients with different needs so the first thing we did was we built out a very lightweight prototype of it where we just packed it together and kind of you know put something out and send it out … as soon as we could to a bunch of different teams with various degrees of expertise and various use cases" - P5

Five participants also did formal usability testing where they observed and measured developers trying out the API:

"We get with our group of developers that build an app and they start working on their APIs, and we measure how long it takes for them to get from 0 to 200… we use that 0 to 200… at different stages. One is during the development side. Two is ... before going into production. And during production" - P22

Here "0 to 200" refers to how long it takes a developer to get a 200 value returned from the API's web server, signaling that the call was processed without error. This metric, also known as Time to Hello World [9], was used by two designers as a measure for how easy it is to use an API. Four designers had done usability studies in the past but found performing studies to be too time and resource expensive to do regularly.

Implications: Best practice requires testing an API with users early, even when the API is not yet implemented or even fully designed. Beta-testing implemented prototype APIs is an existing software engineering practice and should be done. Formal usability testing is rare and also time consuming, but

relying on simple measures like time from 0 to 200 may make usability studies less daunting to perform. In the face of low resources or time, quick feedback from peers *outside* of the API team is a great resource. Teams might try informal user testing like P1's lunchroom exercise, or a hackathon style lunch where developers from inside the company come for food and sit down with members of the API team and follow a "think aloud" protocol [43] where each "user" developer walks the team member developer how they would use the API in its current design. The flexibility of this is that the API can be fully implemented, or just a list of methods on a napkin, and all that matters is observing how users attempting a real task approach your API. Even in these informal settings, however, it is crucial to follow core tenants of formal usability testing so as not to ruin the validity of the exercise, for instance: 1) Predefined tasks for the user to do with your API so that you can later compare how different users respond to the same circumstances. 2) Avoid correcting or overly teaching the user how to use your API when they try it out (even if they mess up) since your API must stand on its own and your real users will not have you sitting behind them. 3) Be open to negative feedback, even if you feel the user doing the task is not knowledgeable or "smart" enough to understand your design (your final API users may very well be the same).

F. API Design Review: Key to Ensuring Quality

All participants reported using design or code reviews as part of their API development process. To help focus the reviews, twenty participants mentioned using automatic tools, like FXCop [39] or Clang-Tidy [40] and custom linters, which identify low-level issues that then do not need to be covered in the review. Instead, they can focus on broader, more subjective issues such as intent, customer workflows, usable naming decisions, and how the code fits consistently with the rest of the company's codebase:

"A manual review should focus on usability and like intangibles about it - about use cases and APIs and then automated tooling should focus on the annoying stuff, like did you name this correctly or you used casing inconsistently, or this is paginated and this isn't paginated" - P18

For most of the companies we talked to, there was a group of API design experts who performed the design reviews, especially for public APIs. Some companies had a small group while others had a much larger group who handled the task. One participant mentioned that their company had a single person act as the expert reviewer for their division, to maintain higher levels of consistency.

Another participant recounted how, before even looking at the API, they would try to understand the problem domain and the resources and relationships that exist within it. After that is understood, the reviewer would check to see how well the API design captured those relationships. However, others noted that reviews failed to serve their purpose when too much high-level design discussion of the API meant they never sat down to concrete code examples that the API's user will have to work with.

The point at which design reviews are introduced into the overall API development process varied widely among participants. For some, the reviews were incorporated as early as when they were sketching out the core abstractions and naming. Five participants strongly encouraged this early review feedback. Others held a different kind of final review of the product with a committee at the end of development.

Implications: Best practice requires having design reviews preferably at multiple stages of the API design. These reviews should be performed including API design experts. There are multiple kinds of review. For instance, in a high level conceptual review, reviewers should consider whether the structure of the API makes sense and accurately reflects the relationships of the problem domain. In a code experience review, the reviewer should try to write or read actual code snippets for a real use case, to review the quality of the source code that must be generated to achieve a user's task.

G. Web and REST APIs

Though a diverse group of designers were interviewed, the majority of participants were involved in web and REST APIs. Some of the usability concerns that were identified were specific to those kinds of APIs. For instance, pagination presented a specific concern for participants in terms of how closely the API's representation of the data should match a consumer-facing UI's representation of the data:

"*The web API might return 10 results per page just like the UI, but if when, then, should the client library do that? Or should the client library return something that looks like an arraylist in Java, call next, and you just get the next one? I haven't seen that actually well handled.*" - P1

Another designer brought up the difficulty in representing data when multiple lists are involved in a response:

"*Pagination is really good when the response has one single list, that needs to be paginated but if it has multiple lists then it becomes more difficult from the service perspective as well as from a customer perspective, to understand where the boundaries are between the two lists and if these two lists are related, that becomes really difficult as well.*" – P21

Designers also had usability concerns regarding the fields contained in the response sent back to users. Determining what information is necessary to cover a range of use cases and what is excessive is a challenge that designers face. Providing too much information could overload the response object sent back to users, but providing too little information means some use cases will not be met. One participant tried filtering to allow users more control regarding what the response contains. Though they mentioned having set patterns for filtering, they also wondered what potentially better ways to structure filters would be. A poorly structured filter could

cause backwards compatibility issues as an API grows in complexity and new users require different information.

Implications: Web and Rest APIs are an enormous area of active API development, but there are still some gaps where there are no obvious design best practices covering how to best chunk and filter data to return to the user. Responses should be designed to help users understand boundaries between relationships in the data that they are requesting.

H. Documentation & User Starting Experience with an API

The initial experience with an API was a major usability concern with seven designers, because from a business perspective, a developer's first encounter with an API largely determines if they will adopt it. If first time users face too many learning barriers, then businesses miss out on customers who attempted to use their products:

"Getting started is something that I have seen as a big challenge for developers starting to use [API X] *because there are lots of concepts involved when it comes to cloud … a lot of these terminologies, cloud-based terminologies and service-specific terminologies. So that is where I have seen the biggest problem or challenge."* - P21

API designers often felt that documentation suffers from discoverability issues. This issue may arise because the names used in the API are more abstract than the use case that a user has in mind - for example a user looking to draw a circle may search documentation for "circle" but the correct query would be 'shape' - or it could occur due to a lack of conceptual knowledge in a domain [44]:

"The [name] we came up with is kind of an analogy... But you just know there are people out there who start typing and hit autocomplete and cross their fingers and it doesn't come up and they just write it themselves. So I think that's always a tough thing is how do we make these things findable." - P11

Examples of high quality API documentation however were brought up when participants discussed APIs they admired, such as the Stripe API which has three panels containing a list of methods, details about those methods, and code snippets that demonstrated common use cases. Not all documentation lives up to the ideal though. Designers admitted that they rely on Stack Overflow posts or community-built tutorials to fill in for gaps in their documentation, and so supplemental material may complement company-provided documentation.

"*I do think developers and API designers treat documentation as an afterthought.*" - P8

Implications: Documentation and support resources are crucial to onboarding a user to the API, and research is needed on optimal ways to display documentation. With new APIs that have the "cold start" problem of no existing support on online communities like Stack Overflow, designers should create example projects and code snippets to demonstrate the API so that new users see how the conceptual pieces of the API come together in concrete use cases.

I. Feedback & Usage are Hard to Measure, Hard to Interpret

Once an API is released, designers and their team highly value feedback to improve the usability and quality of the API:

"*I would like to know where they are being slowed down or points that are particularly frustrating and what parts take a long time to figure out or find*." - P3

Feedback about APIs can be surprisingly hard to gather and interpret. The best case was with internal APIs, where the team had good access to talk to their users directly and with the added benefit of being able to look at their users' code:

"*One nice thing about working at* [Company X] *is that... you can actually just look and see 'okay these are all the places inside* [Company X] *that this API is being called and how it's being called.' You can look at the code, you can get a count of how many places there are and so forth and that has been incredibly useful; it means that my decisions, my thoughts on how things ought to be organized are considerably more informed than they would be otherwise.*" - P14

Designers of public APIs have little direct access to their users and typically had a far larger user base. An exception is large enterprise users of a public API, who more often have a direct form of communication with the team. For web APIs, server-side metrics offered counts about which API were most often used, but only vague clues about use cases or usability:

"*Sometimes a high amount of error codes means that callers do not understand … or sometimes they just don't care and they'd rather just get the error conditions back to check with their call. So it can be difficult to parse out their intents*." - P5

Furthermore, designers said that counts of "how often is this API method called" were often unhelpful to infer real use cases, since it is unknown what the programmer does with the data once they receive it from the API.

For public APIs, there are often ample examples of usage on Github and many programmer questions reported on Stack Overflow, but it can be difficult to identify what is useful. Despite the large amount of data from online programmer communities, six designers we talked to had both actively monitored and failed to gleam useful insights from Stack Overflow and similar places. A major challenge is that there are no tools available for API teams to consume community data in aggregate. One participant had developed a way to mine instances of the API usage from Github repositories, but it should be noted that a custom mining program is not trivial, and still leaves open the problem of aggregating examples to yield insight into where misuse or usability problems may be.

The most reliable source of public feedback was reported to be GitHub issues on public API repositories. Although

designers reported still needing to sift through considerable noise to gleam usable design information, users often post feature requests on GitHub issues which gives designers valuable information and suggestions about future directions for the API. Although there is often an abundance of feature requests, the problem is not so much aggregation. Instead, feature requests lead to healthy debates in the API team about what the API's scope and core design principles should be:

"You know, what's the best API we can design that would serve a reasonable number of people, what would the code look like then? And then how many times does this come up? So you know there are limitless number of feature requests that people ask for, even reasonable things we could think of and we end up having to not add at all for various reasons." - P11

Fourteen designers mentioned other feedback mechanisms like chat channels or customer surveys, but the latter often had too low response sample and too high self-selection issues to yield information about the user population in aggregate.

Implications: Designers of public APIs struggle to get usage feedback after their API is released. Although there are large sources of online programmer community data using a given API, designers currently need better tools to help gather and interpret that data in aggregate. Best practice seems to be to collect anecdotal feedback through Stack Overflow, Github issues, surveys, and direct contact with customers.

J. Automatic Generation of SDKs & Documentation

A web API, on its own, is expressed as textual messages sent to a server. So to improve the usability of web APIs, companies often build Software Development Kits (SDKs) which provide a library wrapper in a certain language for using that API. Interviewees' companies built SDKs for one to as many as nine different languages for a single API. To scale to supporting many languages, some companies use tools that, given a formal specification of the API, will generate the SDK in the various target languages (some also generate documentation). However, generated SDKs were a contentious topic. Interviewees in favor of this approach said that the generated libraries then have consistent naming and abstractions across the SDKs by avoiding idiosyncrasies of individual developers. Interviewees from two companies reported great success with code generation:

"*We did a little bit of investigation into generated SDKs early on and thought they were complete garbage and walked away for two years. The generation technology that we're using today… we wrote our own generator that will generate APIs that are indistinguishable from the hand written APIs*." - P2

Five participants used Swagger API tooling to document an API's design or preview what the API *might* look like in a certain language, but no participant reported using Swagger's code generation tool for their SDKs. Interviewees who avoided generated SDK complained of low quality of generated SDKs or inflexibility for the generator to meet their company's specific requirements (like security).

The two companies that routinely and successfully generated SDKs had: 1) internal custom-made generators that were specific enough to match the company's security policies, style guides, etc., and 2) access to lots of processing power. One participant reported the sheer processing time needed to generate these SDKs limited the number of refinements the team could make.

Since individual languages have vastly different styles and idioms, some participants raised concerns about language specific usability of the generated code. While not impossible, good multi-language usability requires significant work on the generator per language:

"*It's possible to generate SDKs in multiple programming languages from a single model and make them feel idiomatic for each one of those languages you generate for. But what you have to have are experts at each one of those programming languages. We actually have dedicated teams for each language that maintain the code-generator*." - P18

Implications: Technologies to automatically generate SDKs and documentation have greatly improved over the last few years. However, to achieve good usability for different languages requires generation engines carefully tuned by experts in those languages. General-purpose generators like Swagger need to be far more tunable to match the success of bespoke in-house generators, to give designers the freedom to match their company requirements and usability concerns.

VI. LIMITATIONS

This study was limited by the number of participants and companies that they represent, thus may not generalize to all designers or companies. All participants self-selected whether to participate, so participants are primarily designers who have a strong interest in API design quality and API usability.

VII. CONCLUSIONS

Even though there exists some literature, papers, and blogs that talk about API design processes, tools, and guidelines, the interviews that we conducted provide insights into the real world situations and needs. We hope that companies, researchers, and veteran and new API designers can use the information in this paper to improve their own processes, create well-designed APIs, and create new tools and guidelines to help in the design process for future APIs.

ACKNOWLEDGMENTS

This research was supported in part by a grant from Google, and in part by NSF grant CCF-1560137. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the funders.

REFERENCES

[1] B. A. Myers and J. Stylos, “Improving API usability,” *Commun. ACM*, vol. 59, no. 6, pp. 62–69, May 2016.

[2] Keerthi Iyengar, Somesh Khanna, Srinivas Ramadath, Daniel Stephens, “What it really takes to capture the value of APIs,” McKinsey & Company, Sep. 2017.

[3] Press Release From Research, “$200+ Billion Application Programming Interfaces (API) Markets 2017-2022: Focus on Telecoms and Internet of Things,” 07-Sep-2017.

[4] Bala Iyer, Mohan Subramaniam, “The Strategic Value of APIs,” Harvard Business Review, Jan. 2015.

[5] B. A. Myers and J. Stylos, “Improving API usability,” *Commun. ACM*, vol. 59, no. 6, pp. 62–69, 2016.

[6] B. A. Myers, A. J. Ko, T. D. LaToza, and Y. Yoon, “Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools,” *Computer* , vol. 49, no. 7, pp. 44–52, Jul. 2016.

[7] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, “Rethinking SSL development in an appified world,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 49–60.

[8] C. Scaffidi, “Why are APIs difficult to learn and use?,” *Crossroads*, vol. 12, no. 4, pp. 4–4, Aug. 2006.

[9] A. Macvean, L. Church, J. Daughtry, and C. Citro, “API Usability at Scale,” in *27th Annual Workshop of the Psychology of Programming Interest Group-PPIG 2016*, 2016, pp. 177–187.

[10] M. Robillard and R. DeLine, “A field study of API learning obstacles,” *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[11] D. Hou and L. Li, “Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers’ Newsgroup Discussions,” in *2011 IEEE 19th International Conference on Program Comprehension*, 2011, pp. 91–100.

[12] M. Piccioni, C. A. Furia, and B. Meyer, “An Empirical Study of API Usability,” in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013, pp. 5–14.

[13] M. F. Zibran, “What Makes APIs Difficult to Use?,” *International Journal of Computer Science and Network Security*, vol. 8, no. 4, pp. 255–261, 2008.

[14] A. Macvean, M. Maly, and J. Daughtry, “API Design Reviews at Scale,” in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2016, pp. 849–858.

[15] J. Stylos and B. Myers, *Mapping the Space of API Design Decisions*. 2007.

[16] B. Ellis, J. Stylos, and B. Myers, *The Factory Pattern in API Design: A Usability Evaluation*. 2007.

[17] J. Stylos and S. Clarke, *Usability Implications of Requiring Parameters in Objects’ Constructors*. 2007.

[18] J. Stylos and B. A. Myers., “The Implications of Method Placement on API Learnability,” in *Sixteenth ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE 2008)*, 2008, pp. 105–112.

[19] T. Scheller and E. Kuhn, “Automated measurement of API usability: The API Concepts Framework,” *Information and Software Technology*, vol. 61, pp. 145–162, 2015.

[20] G. M. Rama and A. Kak, “Some structural measures of API usability: SOME STRUCTURAL MEASURES OF API USABILITY,” *Softw. Pract. Exp.*, vol. 45, no. 1, pp. 75–110, Jan. 2015.

[21] A. Macvean, L. Church, J. Daughtry, and C. Citro, “API Usability at Scale,” in *27th Annual Workshop of the Psychology of Programming Interest Group - PPIG 2016*, 2016, pp. 177–187.

[22] Google, “API Design Guide,” 21-Feb-2017. [Online]. Available: https://cloud.google.com/apis/design/. [Accessed: 2017].

[23] Microsoft, “API design,” 13-Jul-2016. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design. [Accessed: 2017].

[24] L. Murphy, T. Alliyu, M. B. Kery, A. Macvean, B. A. Myers, “Preliminary Analysis of REST API Style Guidelines,” in *8th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU’2017) at SPLASH 2017*, p. to appear.

[25] K. Cwalina and B. Abrams, *Framework Design Guidelines, Conventions, Idioms, and Patterns for Reusable .NET Libraries*. Upper-Saddle River, NJ: Addison-Wesley, 2006.

[26] J. Bloch, *Effective Java Programming Language Guide*. Mountain View, CA: Sun Microsystems, 2001.

[27] S. Clarke, *Describing and Measuring API Usability with the Cognitive Dimensions*. 2005.

[28] A. Macvean, M. Maly, and J. Daughtry, “API Design Reviews at Scale,” in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA ’16)*, 2016, pp. 849–858.

[29] J. Stylos and B. A. Myers., “The Implications of Method Placement on API Learnability,” in *Sixteenth ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE 2008)*, 2008, pp. 105–112.

[30] T. Grill, O. Polacek, and M. Tscheligi, “Methods towards API Usability: A Structural Analysis of Usability Problem Categories,” in *Human-Centered Software Engineering*, vol. 7623, Winckler, Marco, and E. Al, Eds. Toulouse, France: Springer Berlin Heidelberg, 2012, pp. 164–180.

[31] U. Farooq, L. Welicki, and D. Zirkler, “API usability peer reviews,” in *Proceedings of the 28th international conference on Human factors in computing systems - CHI ’10*, 2010.

[32] M. Henning, “API Design Matters,” *ACM Queue*, vol. 5, no. 4, pp. 24–36, 2007.

[33] I. Rus and M. Lindvall, “Knowledge management in software engineering,” *IEEE Softw.*, vol. 19, no. 3, pp. 26–38, 2002.

[34] E. Bjarnason, K. Smolander, E. Engström, and P. Runeson, “A theory of distances in software engineering,” *Information and Software Technology*, vol. 70, pp. 204–219, Feb. 2016.

[35] H. K. Edwards and V. Sridhar, “Analysis of the effectiveness of global virtual teams in software engineering projects,” in *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, 2003.

[36] J. Corbin and A. Strauss, “Grounded Theory Research: Procedures, Canons and Evaluative Criteria,” *Zeitschrift für Soziologie*, vol. 19, no. 6, p. 515, Jan. 1990.

[37] M. L. McHugh, “Interrater reliability: the kappa statistic,” *Biochem. Med.* , vol. 22, no. 3, pp. 276–282, 2012.

[38] C. Sadowski, J. van Gogh, C. Jaspan, E. Soderberg, and C. Winter, “Tricorder: Building a Program Analysis Ecosystem,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015.

[39] FxCop. FxCop, 2018. https://msdn.microsoft.com/en-us/library/bb429476.aspx [Accessed: 2018]

[40] Clang. Clang-Tidy, 2018. http://clang.llvm.org/extra/clang-tidy/ [Accessed: 2018]

[41] E. Murphy-Hill, C. Sadowski, A. Head, J.Daughtry, A.Macvean, C. Jaspan, & C. “Discovering API Usability Problems at Scale.” in *Proceedings of the 2nd International Workshop on API Usage and Evolution* (2018), pp. 14-17.

[42] V. Braun, & V. Clarke. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, *3*(2), 77-101.

[43] C. Lewis, & J. Rieman. (1993). Task-centered user interface design. *A Practical Introduction*.

[44] A. J. Ko, & Y. Riche. (2011, September). The role of conceptual knowledge in API usability. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on* (pp. 173-176). IEEE.